        HTTP Authentication: Basic and Digest Access Authentication

Status of this Memo

Copyright Notice

Abstract

    "HTTP/1.0", includes the specification for a Basic Access
    Authentication scheme. This scheme is not considered to be a secure
    method of user authentication (unless used in conjunction with some
    external secure system such as SSL [5]), as the user name and
    password are passed over the network as cleartext.

    This document also provides the specification for HTTP's
    authentication framework, the original Basic authentication scheme
    and a scheme based on cryptographic hashes, referred to as "Digest
    Access Authentication". It is therefore also intended to serve as a
    replacement for RFC 2069 [6]. Some optional elements specified by
    RFC 2069 have been removed from this specification due to problems
    found since its publication; other new elements have been added for
    compatibility, those new elements have been made optional, but are
    strongly recommended.

Franks, et al.   Standards Track   [Page 1]

RFC 2617   HTTP Authentication   June 1999

    Like Basic, Digest access authentication verifies that both parties
    to a communication know a shared secret (a password); unlike Basic,
    this verification can be done without sending the password in the
    clear, which is Basic's biggest weakness. As with most other
    authentication protocols, the greatest sources of risks are usually
    found not in the core protocol itself but in policies and procedures
    surrounding its use.

Table of Contents

Franks, et al.   Standards Track   [Page 2]

RFC 2617   HTTP Authentication   June 1999

1 Access Authentication

1.1 Reliance on the HTTP/1.1 Specification

    This specification is a companion to the HTTP/1.1 specification [2].
    It uses the augmented BNF section 2.1 of that document, and relies on
    both the non-terminals defined in that document and other aspects of
    the HTTP/1.1 specification.

1.2 Access Authentication Framework

    HTTP provides a simple challenge-response authentication mechanism
    that MAY be used by a server to challenge a client request and by a
    client to provide authentication information. It uses an extensible,
    case-insensitive token to identify the authentication scheme,
    followed by a comma-separated list of attribute-value pairs which
    carry the parameters necessary for achieving authentication via that
    scheme.

        auth-scheme   = token
        auth-param   = token "=" (token | quoted-string)

    The 401 (Unauthorized) response message is used by an origin server
    to challenge the authorization of a user agent. This response MUST
    include a WWW-Authenticate header field containing at least one

challenge applicable to the requested resource. The 407 (Proxy
Authentication Required) response message is used by a proxy to
challenge the authorization of a client and MUST include a Proxy-
Authenticate header field containing at least one challenge
applicable to the proxy for the requested resource.

    challenge   = auth-scheme 1*SP 1#auth-param

Note: User agents will need to take special care in parsing the WWW-
Authenticate or Proxy-Authenticate header field value if it contains
more than one challenge, or if more than one WWW-Authenticate header
field is provided, since the contents of a challenge may itself
contain a comma-separated list of authentication parameters.

The authentication parameter realm is defined for all authentication
schemes:

    realm   = "realm" "=" realm-value
    realm-value = quoted-string.

Franks, et al.   Standards Track   [Page 3]

RFC 2617   HTTP Authentication   June 1999

The realm directive (case-insensitive) is required for all
authentication schemes that issue a challenge. The realm value
(case-sensitive), in combination with the canonical root URL (the
absoluteURI for the server whose abs_path is empty; see section 5.1.2
of [2]) of the server being accessed, defines the protection space.
These realms allow the protected resources on a server to be
partitioned into a set of protection spaces, each with its own
authentication scheme and/or authorization database. The realm value
is a string, generally assigned by the origin server, which may have
additional semantics specific to the authentication scheme. Note that
there may be multiple challenges with the same auth-scheme but
different realms.

A user agent that wishes to authenticate itself with an origin
server--usually, but not necessarily, after receiving a 401
(Unauthorized)--MAY do so by including an Authorization header field
with the request. A client that wishes to authenticate itself with a
proxy--usually, but not necessarily, after receiving a 407 (Proxy
Authentication Required)--MAY do so by including a Proxy-
Authorization header field with the request. Both the Authorization
field value and the Proxy-Authorization field value consist of
credentials containing the authentication information of the client
for the realm of the resource being requested. The user agent MUST
choose to use one of the challenges with the strongest auth-scheme it
understands and request credentials from the user based upon that
challenge.

credentials = auth-scheme #auth-param

    Note that many browsers will only recognize Basic and will require
    that it be the first auth-scheme presented. Servers should only
    include Basic if it is minimally acceptable.

The protection space determines the domain over which credentials can
be automatically applied. If a prior request has been authorized, the
same credentials MAY be reused for all other requests within that
protection space for a period of time determined by the
authentication scheme, parameters, and/or user preference. Unless
otherwise defined by the authentication scheme, a single protection
space cannot extend outside the scope of its server.

If the origin server does not wish to accept the credentials sent
with a request, it SHOULD return a 401 (Unauthorized) response. The
response MUST include a WWW-Authenticate header field containing at
least one (possibly new) challenge applicable to the requested
resource. If a proxy does not accept the credentials sent with a
request, it SHOULD return a 407 (Proxy Authentication Required). The
response MUST include a Proxy-Authenticate header field containing a

RFC 2617   HTTP Authentication   June 1999

    (possibly new) challenge applicable to the proxy for the requested
    resource.

    The HTTP protocol does not restrict applications to this simple
    challenge-response mechanism for access authentication. Additional
    mechanisms MAY be used, such as encryption at the transport level or
    via message encapsulation, and with additional header fields
    specifying authentication information. However, these additional
    mechanisms are not defined by this specification.

    Proxies MUST be completely transparent regarding user agent
    authentication by origin servers. That is, they must forward the
    WWW-Authenticate and Authorization headers untouched, and follow the
    rules found in section 14.8 of [2]. Both the Proxy-Authenticate and
    the Proxy-Authorization header fields are hop-by-hop headers (see
    section 13.5.1 of [2]).

2 Basic Authentication Scheme

    The "basic" authentication scheme is based on the model that the
    client must authenticate itself with a user-ID and a password for
    each realm. The realm value should be considered an opaque string
    which can only be compared for equality with other realms on that
    server. The server will service the request only if it can validate
    the user-ID and password for the protection space of the Request-URI.
    There are no optional authentication parameters.

    For Basic, the framework above is utilized as follows:

        challenge   = "Basic" realm
        credentials = "Basic" basic-credentials

    Upon receipt of an unauthorized request for a URI within the
    protection space, the origin server MAY respond with a challenge like
    the following:

        WWW-Authenticate: Basic realm="WallyWorld"

    where "WallyWorld" is the string assigned by the server to identify
    the protection space of the Request-URI. A proxy may respond with the
    same challenge using the Proxy-Authenticate header field.

    To receive authorization, the client sends the userid and password,
    separated by a single colon (":") character, within a base64 [7]
    encoded string in the credentials.

        basic-credentials = base64-user-pass
        base64-user-pass = <base64 [4] encoding of user-pass,

RFC 2617   HTTP Authentication   June 1999

                        except not limited to 76 char/line>
        user-pass   = userid ":" password
        userid   = *<TEXT excluding ":">
        password   = *TEXT

    Userids might be case sensitive.

    If the user agent wishes to send the userid "Aladdin" and password

"open sesame", it would use the following header field:

    Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==

A client SHOULD assume that all paths at or deeper than the depth of
the last symbolic element in the path field of the Request-URI also
are within the protection space specified by the Basic realm value of
the current challenge. A client MAY preemptively send the
corresponding Authorization header with requests for resources in
that space without receipt of another challenge from the server.
Similarly, when a client sends a request to a proxy, it may reuse a
userid and password in the Proxy-Authorization header field without
receiving another challenge from the proxy server. See section 4 for
security considerations associated with Basic authentication.

3 Digest Access Authentication Scheme

3.1 Introduction

3.1.1 Purpose

The protocol referred to as "HTTP/1.0" includes the specification for
a Basic Access Authentication scheme[1]. That scheme is not
considered to be a secure method of user authentication, as the user
name and password are passed over the network in an unencrypted form.
This section provides the specification for a scheme that does not
send the password in cleartext, referred to as "Digest Access
Authentication".

The Digest Access Authentication scheme is not intended to be a
complete answer to the need for security in the World Wide Web. This
scheme provides no encryption of message content. The intent is
simply to create an access authentication method that avoids the most
serious flaws of Basic authentication.

3.1.2 Overall Operation

Like Basic Access Authentication, the Digest scheme is based on a
simple challenge-response paradigm. The Digest scheme challenges
using a nonce value. A valid response contains a checksum (by

Franks, et al.   Standards Track   [Page 6]

RFC 2617   HTTP Authentication   June 1999

default, the MD5 checksum) of the username, the password, the given
nonce value, the HTTP method, and the requested URI. In this way, the
password is never sent in the clear. Just as with the Basic scheme,
the username and password must be prearranged in some fashion not
addressed by this document.

3.1.3 Representation of digest values

An optional header allows the server to specify the algorithm used to
create the checksum or digest. By default the MD5 algorithm is used
and that is the only algorithm described in this document.

For the purposes of this document, an MD5 digest of 128 bits is
represented as 32 ASCII printable characters. The bits in the 128 bit
digest are converted from most significant to least significant bit,
four bits at a time to their ASCII presentation as follows. Each four
bits is represented by its familiar hexadecimal notation from the
characters 0123456789abcdef. That is, binary 0000 gets represented by
the character '0', 0001, by '1', and so on up to the representation
of 1111 as 'f'.

3.1.4 Limitations

The Digest authentication scheme described in this document suffers
from many known limitations. It is intended as a replacement for
Basic authentication and nothing more. It is a password-based system

and (on the server side) suffers from all the same problems of any
password system. In particular, no provision is made in this protocol
for the initial secure arrangement between user and server to
establish the user's password.

Users and implementors should be aware that this protocol is not as
secure as Kerberos, and not as secure as any client-side private-key
scheme. Nevertheless it is better than nothing, better than what is
commonly used with telnet and ftp, and better than Basic
authentication.

3.2 Specification of Digest Headers

The Digest Access Authentication scheme is conceptually similar to
the Basic scheme. The formats of the modified WWW-Authenticate header
line and the Authorization header line are specified below. In
addition, a new header, Authentication-Info, is specified.

Franks, et al.   Standards Track   [Page 7]

RFC 2617   HTTP Authentication   June 1999

3.2.1 The WWW-Authenticate Response Header

If a server receives a request for an access-protected object, and an
acceptable Authorization header is not sent, the server responds with
a "401 Unauthorized" status code, and a WWW-Authenticate header as
per the framework defined above, which for the digest scheme is
utilized as follows:

```
        challenge        =  "Digest" digest-challenge

        digest-challenge = 1#( realm | [ domain ] | nonce |
                           [ opaque ] |[ stale ] | [ algorithm ] |
                           [ qop-options ] | [auth-param] )


        domain           = "domain" "=" <"> URI ( 1*SP URI ) <">
        URI              = absoluteURI | abs_path
        nonce            = "nonce" "=" nonce-value
        nonce-value      = quoted-string
        opaque           = "opaque" "=" quoted-string
        stale            = "stale" "=" ( "true" | "false" )
        algorithm        = "algorithm" "=" ( "MD5" | "MD5-sess" |
                             token )
        qop-options      = "qop" "=" <"> 1#qop-value <">
        qop-value        = "auth" | "auth-int" | token
```

The meanings of the values of the directives used above are as
follows:

realm
    A string to be displayed to users so they know which username and
    password to use. This string should contain at least the name of
    the host performing the authentication and might additionally
    indicate the collection of users who might have access. An example
    might be "registered_users@gotham.news.com".

domain
    A quoted, space-separated list of URIs, as specified in RFC XURI
    [7], that define the protection space. If a URI is an abs_path, it
    is relative to the canonical root URL (see section 1.2 above) of
    the server being accessed. An absoluteURI in this list may refer to
    a different server than the one being accessed. The client can use
    this list to determine the set of URIs for which the same

authentication information may be sent: any URI that has a URI in
this list as a prefix (after both have been made absolute) may be
assumed to be in the same protection space. If this directive is
omitted or its value is empty, the client should assume that the
protection space consists of all URIs on the responding server.

Franks, et al.   Standards Track   [Page 8]

RFC 2617   HTTP Authentication   June 1999

This directive is not meaningful in Proxy-Authenticate headers, for
which the protection space is always the entire proxy; if present
it should be ignored.

nonce
A server-specified data string which should be uniquely generated
each time a 401 response is made. It is recommended that this
string be base64 or hexadecimal data. Specifically, since the
string is passed in the header lines as a quoted string, the
double-quote character is not allowed.

The contents of the nonce are implementation dependent. The quality
of the implementation depends on a good choice. A nonce might, for
example, be constructed as the base 64 encoding of

    time-stamp H(time-stamp ":" ETag ":" private-key)

where time-stamp is a server-generated time or other non-repeating
value, ETag is the value of the HTTP ETag header associated with
the requested entity, and private-key is data known only to the
server. With a nonce of this form a server would recalculate the
hash portion after receiving the client authentication header and
reject the request if it did not match the nonce from that header
or if the time-stamp value is not recent enough. In this way the
server can limit the time of the nonce's validity. The inclusion of
the ETag prevents a replay request for an updated version of the
resource. (Note: including the IP address of the client in the
nonce would appear to offer the server the ability to limit the
reuse of the nonce to the same client that originally got it.
However, that would break proxy farms, where requests from a single
user often go through different proxies in the farm. Also, IP
address spoofing is not that hard.)

An implementation might choose not to accept a previously used
nonce or a previously used digest, in order to protect against a
replay attack. Or, an implementation might choose to use one-time
nonces or digests for POST or PUT requests and a time-stamp for GET
requests. For more details on the issues involved see section 4.
of this document.

The nonce is opaque to the client.

opaque
A string of data, specified by the server, which should be returned
by the client unchanged in the Authorization header of subsequent
requests with URIs in the same protection space. It is recommended
that this string be base64 or hexadecimal data.

Franks, et al.   Standards Track   [Page 9]

RFC 2617   HTTP Authentication   June 1999

stale
A flag, indicating that the previous request from the client was
rejected because the nonce value was stale. If stale is TRUE

(case-insensitive), the client may wish to simply retry the request
with a new encrypted response, without reprompting the user for a
new username and password. The server should only set stale to TRUE
if it receives a request for which the nonce is invalid but with a
valid digest for that nonce (indicating that the client knows the
correct username/password). If stale is FALSE, or anything other
than TRUE, or the stale directive is not present, the username
and/or password are invalid, and new values must be obtained.

algorithm
   A string indicating a pair of algorithms used to produce the digest
   and a checksum. If this is not present it is assumed to be "MD5".
   If the algorithm is not understood, the challenge should be ignored
   (and a different one used, if there is more than one).

   In this document the string obtained by applying the digest
   algorithm to the data "data" with secret "secret" will be denoted
   by KD(secret, data), and the string obtained by applying the
   checksum algorithm to the data "data" will be denoted H(data). The
   notation unq(X) means the value of the quoted-string X without the
   surrounding quotes.

   For the "MD5" and "MD5-sess" algorithms

       H(data) = MD5(data)

   and

       KD(secret, data) = H(concat(secret, ":", data))

   i.e., the digest is the MD5 of the secret concatenated with a colon
   concatenated with the data. The "MD5-sess" algorithm is intended to
   allow efficient 3rd party authentication servers; for the
   difference in usage, see the description in section 3.2.2.2.

qop-options
   This directive is optional, but is made so only for backward
   compatibility with RFC 2069 [6]; it SHOULD be used by all
   implementations compliant with this version of the Digest scheme.
   If present, it is a quoted string of one or more tokens indicating
   the "quality of protection" values supported by the server. The
   value "auth" indicates authentication; the value "auth-int"
   indicates authentication with integrity protection; see the

Franks, et al.   Standards Track   [Page 10]

RFC 2617   HTTP Authentication   June 1999

   descriptions below for calculating the response directive value for
   the application of this choice. Unrecognized options MUST be
   ignored.

auth-param
   This directive allows for future extensions. Any unrecognized
   directive MUST be ignored.

3.2.2 The Authorization Request Header

   The client is expected to retry the request, passing an Authorization
   header line, which is defined according to the framework above,
   utilized as follows.

       credentials   = "Digest" digest-response
       digest-response = 1#(username | realm | nonce | digest-uri
                          | response | [algorithm] | [cnonce] |
                          [opaque] | [message-qop] |
                              [nonce-count] | [auth-param])

```
username              = "username" "=" username-value
username-value        = quoted-string
digest-uri            = "uri" "=" digest-uri-value
digest-uri-value      = request-uri   ; As specified by HTTP/1.1
message-qop           = "qop" "=" qop-value
cnonce                = "cnonce" "=" cnonce-value
cnonce-value          = nonce-value
nonce-count           = "nc" "=" nc-value
nc-value              = 8LHEX
response              = "response" "=" request-digest
request-digest = <"> 32LHEX <">
LHEX                  =  "0" | "1" | "2" | "3" |
                         "4" | "5" | "6" | "7" |
                         "8" | "9" | "a" | "b" |
                         "c" | "d" | "e" | "f"
```

The values of the opaque and algorithm fields must be those supplied
in the WWW-Authenticate response header for the entity being
requested.

response
   A string of 32 hex digits computed as defined below, which proves
   that the user knows a password

username
   The user's name in the specified realm.

Franks, et al.   Standards Track   [Page 11]

RFC 2617   HTTP Authentication   June 1999

digest-uri
   The URI from Request-URI of the Request-Line; duplicated here
   because proxies are allowed to change the Request-Line in transit.

qop
   Indicates what "quality of protection" the client has applied to
   the message. If present, its value MUST be one of the alternatives
   the server indicated it supports in the WWW-Authenticate header.
   These values affect the computation of the request-digest. Note
   that this is a single token, not a quoted list of alternatives as
   in WWW- Authenticate. This directive is optional in order to
   preserve backward compatibility with a minimal implementation of
   RFC 2069 [6], but SHOULD be used if the server indicated that qop
   is supported by providing a qop directive in the WWW-Authenticate
   header field.

cnonce
   This MUST be specified if a qop directive is sent (see above), and
   MUST NOT be specified if the server did not send a qop directive in
   the WWW-Authenticate header field. The cnonce-value is an opaque
   quoted string value provided by the client and used by both client
   and server to avoid chosen plaintext attacks, to provide mutual
   authentication, and to provide some message integrity protection.
   See the descriptions below of the calculation of the response-
   digest and request-digest values.

nonce-count
   This MUST be specified if a qop directive is sent (see above), and
   MUST NOT be specified if the server did not send a qop directive in
   the WWW-Authenticate header field. The nc-value is the hexadecimal
   count of the number of requests (including the current request)
   that the client has sent with the nonce value in this request. For
   example, in the first request sent in response to a given nonce
   value, the client sends "nc=00000001". The purpose of this
   directive is to allow the server to detect request replays by
   maintaining its own copy of this count - if the same nc-value is
   seen twice, then the request is a replay.   See the description
   below of the construction of the request-digest value.

auth-param
    This directive allows for future extensions. Any unrecognized
    directive MUST be ignored.

If a directive or its value is improper, or required directives are
missing, the proper response is 400 Bad Request. If the request-
digest is invalid, then a login failure should be logged, since
repeated login failures from a single client may indicate an attacker
attempting to guess passwords.

Franks, et al.   Standards Track   [Page 12]

RFC 2617   HTTP Authentication   June 1999

    The definition of request-digest above indicates the encoding for its
    value. The following definitions show how the value is computed.

3.2.2.1 Request-Digest

    If the "qop" value is "auth" or "auth-int":

        request-digest = <"> < KD (H(A1),   unq(nonce-value)
                                            ":" nc-value
                                            ":" unq(cnonce-value)
                                            ":" unq(qop-value)
                                            ":" H(A2)
) <">

    If the "qop" directive is not present (this construction is for
    compatibility with RFC 2069):

        request-digest =
                    <"> < KD (H(A1), unq(nonce-value) ":" H(A2)) >
    <">

    See below for the definitions for A1 and A2.

3.2.2.2 A1

    If the "algorithm" directive's value is "MD5" or is unspecified, then
    A1 is:

        A1   = unq(username-value) ":" unq(realm-value) ":" passwd

    where

        passwd   = < user's password >

    If the "algorithm" directive's value is "MD5-sess", then A1 is
    calculated only once - on the first request by the client following
    receipt of a WWW-Authenticate challenge from the server. It uses the
    server nonce from that challenge, and the first client nonce value to
    construct A1 as follows:

        A1   = H(unq(username-value) ":" unq(realm-value)
                    ":" passwd)
                    ":" unq(nonce-value) ":" unq(cnonce-value)

    This creates a 'session key' for the authentication of subsequent
    requests and responses which is different for each "authentication
    session", thus limiting the amount of material hashed with any one
    key. (Note: see further discussion of the authentication session in

Franks, et al.   Standards Track   [Page 13]

RFC 2617   HTTP Authentication   June 1999

section 3.3.) Because the server need only use the hash of the user
credentials in order to create the A1 value, this construction could
be used in conjunction with a third party authentication service so
that the web server would not need the actual password value. The
specification of such a protocol is beyond the scope of this
specification.

3.2.2.3 A2

If the "qop" directive's value is "auth" or is unspecified, then A2
is:

    A2   = Method ":" digest-uri-value

If the "qop" value is "auth-int", then A2 is:

    A2   = Method ":" digest-uri-value ":" H(entity-body)

3.2.2.4 Directive values and quoted-string

Note that the value of many of the directives, such as "username-
value", are defined as a "quoted-string". However, the "unq" notation
indicates that surrounding quotation marks are removed in forming the
string A1. Thus if the Authorization header includes the fields

    username="Mufasa", realm=myhost@testrealm.com

and the user Mufasa has password "Circle Of Life" then H(A1) would be
H(Mufasa:myhost@testrealm.com:Circle Of Life) with no quotation marks
in the digested string.

No white space is allowed in any of the strings to which the digest
function H() is applied unless that white space exists in the quoted
strings or entity body whose contents make up the string to be
digested. For example, the string A1 illustrated above must be

    Mufasa:myhost@testrealm.com:Circle Of Life

with no white space on either side of the colons, but with the white
space between the words used in the password value. Likewise, the
other strings digested by H() must not have white space on either
side of the colons which delimit their fields unless that white space
was in the quoted strings or entity body being digested.

Also note that if integrity protection is applied (qop=auth-int), the
H(entity-body) is the hash of the entity body, not the message body -
it is computed before any transfer encoding is applied by the sender

Franks, et al.   Standards Track   [Page 14]

RFC 2617   HTTP Authentication   June 1999

and after it has been removed by the recipient. Note that this
includes multipart boundaries and embedded headers in each part of
any multipart content-type.

3.2.2.5 Various considerations

The "Method" value is the HTTP request method as specified in section
5.1.1 of [2]. The "request-uri" value is the Request-URI from the
request line as specified in section 5.1.2 of [2]. This may be "*",
an "absoluteURL" or an "abs_path" as specified in section 5.1.2 of
[2], but it MUST agree with the Request-URI. In particular, it MUST
be an "absoluteURL" if the Request-URI is an "absoluteURL". The
"cnonce-value" is an optional client-chosen value whose purpose is
to foil chosen plaintext attacks.

The authenticating server must assure that the resource designated by

the "uri" directive is the same as the resource specified in the
Request-Line; if they are not, the server SHOULD return a 400 Bad
Request error. (Since this may be a symptom of an attack, server
implementers may want to consider logging such errors.) The purpose
of duplicating information from the request URL in this field is to
deal with the possibility that an intermediate proxy may alter the
client's Request-Line. This altered (but presumably semantically
equivalent) request would not result in the same digest as that
calculated by the client.

Implementers should be aware of how authenticated transactions
interact with shared caches. The HTTP/1.1 protocol specifies that
when a shared cache (see section 13.7 of [2]) has received a request
containing an Authorization header and a response from relaying that
request, it MUST NOT return that response as a reply to any other
request, unless one of two Cache-Control (see section 14.9 of [2])
directives was present in the response. If the original response
included the "must-revalidate" Cache-Control directive, the cache MAY
use the entity of that response in replying to a subsequent request,
but MUST first revalidate it with the origin server, using the
request headers from the new request to allow the origin server to
authenticate the new request. Alternatively, if the original response
included the "public" Cache-Control directive, the response entity
MAY be returned in reply to any subsequent request.

3.2.3 The Authentication-Info Header

The Authentication-Info header is used by the server to communicate
some information regarding the successful authentication in the
response.

Franks, et al.   Standards Track   [Page 15]

RFC 2617   HTTP Authentication   June 1999

        AuthenticationInfo = "Authentication-Info" ":" auth-info
        auth-info   = 1#(nextnonce | [message-qop]
                               | [response-auth] | [cnonce]
                               | [nonce-count])
        nextnonce   = "nextnonce" "=" nonce-value
        response-auth   = "rspauth" "=" response-digest
        response-digest   = <"> *LHEX <">

    The value of the nextnonce directive is the nonce the server wishes
    the client to use for a future authentication response. The server
    may send the Authentication-Info header with a nextnonce field as a
    means of implementing one-time or otherwise changing nonces. If the
    nextnonce field is present the client SHOULD use it when constructing
    the Authorization header for its next request. Failure of the client
    to do so may result in a request to re-authenticate from the server
    with the "stale=TRUE".

        Server implementations should carefully consider the performance
        implications of the use of this mechanism; pipelined requests will
        not be possible if every response includes a nextnonce directive
        that must be used on the next request received by the server.
        Consideration should be given to the performance vs. security
        tradeoffs of allowing an old nonce value to be used for a limited
        time to permit request pipelining. Use of the nonce-count can
        retain most of the security advantages of a new server nonce
        without the deleterious affects on pipelining.

    message-qop
        Indicates the "quality of protection" options applied to the
        response by the server. The value "auth" indicates authentication;
        the value "auth-int" indicates authentication with integrity
        protection. The server SHOULD use the same value for the message-
        qop directive in the response as was sent by the client in the
        corresponding request.

The optional response digest in the "response-auth" directive
supports mutual authentication -- the server proves that it knows the
user's secret, and with qop=auth-int also provides limited integrity
protection of the response. The "response-digest" value is calculated
as for the "request-digest" in the Authorization header, except that
if "qop=auth" or is not specified in the Authorization header for the
request, A2 is

    A2   = ":" digest-uri-value

and if "qop=auth-int", then A2 is

    A2   = ":" digest-uri-value ":" H(entity-body)

Franks, et al.   Standards Track   [Page 16]

RFC 2617   HTTP Authentication   June 1999

where "digest-uri-value" is the value of the "uri" directive on the
Authorization header in the request. The "cnonce-value" and "nc-
value" MUST be the ones for the client request to which this message
is the response. The "response-auth", "cnonce", and "nonce-count"
directives MUST BE present if "qop=auth" or "qop=auth-int" is
specified.

The Authentication-Info header is allowed in the trailer of an HTTP
message transferred via chunked transfer-coding.

3.3 Digest Operation

Upon receiving the Authorization header, the server may check its
validity by looking up the password that corresponds to the submitted
username. Then, the server must perform the same digest operation
(e.g., MD5) performed by the client, and compare the result to the
given request-digest value.

Note that the HTTP server does not actually need to know the user's
cleartext password. As long as H(A1) is available to the server, the
validity of an Authorization header may be verified.

The client response to a WWW-Authenticate challenge for a protection
space starts an authentication session with that protection space.
The authentication session lasts until the client receives another
WWW-Authenticate challenge from any server in the protection space. A
client should remember the username, password, nonce, nonce count and
opaque values associated with an authentication session to use to
construct the Authorization header in future requests within that
protection space. The Authorization header may be included
preemptively; doing so improves server efficiency and avoids extra
round trips for authentication challenges. The server may choose to
accept the old Authorization header information, even though the
nonce value included might not be fresh. Alternatively, the server
may return a 401 response with a new nonce value, causing the client
to retry the request; by specifying stale=TRUE with this response,
the server tells the client to retry with the new nonce, but without
prompting for a new username and password.

Because the client is required to return the value of the opaque
directive given to it by the server for the duration of a session,
the opaque data may be used to transport authentication session state
information. (Note that any such use can also be accomplished more
easily and safely by including the state in the nonce.) For example,
a server could be responsible for authenticating content that
actually sits on another server. It would achieve this by having the
first 401 response include a domain directive whose value includes a
URI on the second server, and an opaque directive whose value

Franks, et al.   Standards Track   [Page 17]

contains the state information. The client will retry the request, at
which time the server might respond with a 301/302 redirection,
pointing to the URI on the second server. The client will follow the
redirection, and pass an Authorization header , including the
<opaque> data.

As with the basic scheme, proxies must be completely transparent in
the Digest access authentication scheme. That is, they must forward
the WWW-Authenticate, Authentication-Info and Authorization headers
untouched. If a proxy wants to authenticate a client before a request
is forwarded to the server, it can be done using the Proxy-
Authenticate and Proxy-Authorization headers described in section 3.6
below.

3.4 Security Protocol Negotiation

It is useful for a server to be able to know which security schemes a
client is capable of handling.

It is possible that a server may want to require Digest as its
authentication method, even if the server does not know that the
client supports it. A client is encouraged to fail gracefully if the
server specifies only authentication schemes it cannot handle.

3.5 Example

The following example assumes that an access-protected document is
being requested from the server via a GET request. The URI of the
document is "http://www.nowhere.org/dir/index.html". Both client and
server know that the username for this document is "Mufasa", and the
password is "Circle Of Life" (with one space between each of the
three words).

The first time the client requests the document, no Authorization
header is sent, so the server responds with:

```
    HTTP/1.1 401 Unauthorized
    WWW-Authenticate: Digest
            realm="testrealm@host.com",
            qop="auth,auth-int",
            nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
            opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

The client may prompt the user for the username and password, after
which it will respond with a new request, including the following
Authorization header:

```
    Authorization: Digest username="Mufasa",
            realm="testrealm@host.com",
            nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
            uri="/dir/index.html",
            qop=auth,
            nc=00000001,
            cnonce="0a4f113b",
            response="6629fae49393a05397450978507c4ef1",
            opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

3.6 Proxy-Authentication and Proxy-Authorization

The digest authentication scheme may also be used for authenticating
users to proxies, proxies to proxies, or proxies to origin servers by

use of the Proxy-Authenticate and Proxy-Authorization headers. These
headers are instances of the Proxy-Authenticate and Proxy-
Authorization headers specified in sections 10.33 and 10.34 of the
HTTP/1.1 specification [2] and their behavior is subject to
restrictions described there. The transactions for proxy
authentication are very similar to those already described. Upon
receiving a request which requires authentication, the proxy/server
must issue the "407 Proxy Authentication Required" response with a
"Proxy-Authenticate" header. The digest-challenge used in the
Proxy-Authenticate header is the same as that for the WWW-
Authenticate header as defined above in section 3.2.1.

The client/proxy must then re-issue the request with a Proxy-
Authorization header, with directives as specified for the
Authorization header in section 3.2.2 above.

On subsequent responses, the server sends Proxy-Authentication-Info
with directives the same as those for the Authentication-Info header
field.

Note that in principle a client could be asked to authenticate itself
to both a proxy and an end-server, but never in the same response.

4 Security Considerations

4.1 Authentication of Clients using Basic Authentication

The Basic authentication scheme is not a secure method of user
authentication, nor does it in any way protect the entity, which is
transmitted in cleartext across the physical network used as the
carrier. HTTP does not prevent additional authentication schemes and
encryption mechanisms from being employed to increase security or the
addition of enhancements (such as schemes to use one-time passwords)
to Basic authentication.

Franks, et al.   Standards Track   [Page 19]

RFC 2617   HTTP Authentication   June 1999

The most serious flaw in Basic authentication is that it results in
the essentially cleartext transmission of the user's password over
the physical network. It is this problem which Digest Authentication
attempts to address.

Because Basic authentication involves the cleartext transmission of
passwords it SHOULD NOT be used (without enhancements) to protect
sensitive or valuable information.

A common use of Basic authentication is for identification purposes
-- requiring the user to provide a user name and password as a means
of identification, for example, for purposes of gathering accurate
usage statistics on a server. When used in this way it is tempting to
think that there is no danger in its use if illicit access to the
protected documents is not a major concern. This is only correct if
the server issues both user name and password to the users and in
particular does not allow the user to choose his or her own password.
The danger arises because naive users frequently reuse a single
password to avoid the task of maintaining multiple passwords.

If a server permits users to select their own passwords, then the
threat is not only unauthorized access to documents on the server but
also unauthorized access to any other resources on other systems that
the user protects with the same password. Furthermore, in the
server's password database, many of the passwords may also be users'
passwords for other sites. The owner or administrator of such a
system could therefore expose all users of the system to the risk of
unauthorized access to all those sites if this information is not
maintained in a secure fashion.

Basic Authentication is also vulnerable to spoofing by counterfeit
servers. If a user can be led to believe that he is connecting to a

host containing information protected by Basic authentication when,
in fact, he is connecting to a hostile server or gateway, then the
attacker can request a password, store it for later use, and feign an
error. This type of attack is not possible with Digest
Authentication. Server implementers SHOULD guard against the
possibility of this sort of counterfeiting by gateways or CGI
scripts. In particular it is very dangerous for a server to simply
turn over a connection to a gateway. That gateway can then use the
persistent connection mechanism to engage in multiple transactions
with the client while impersonating the original server in a way that
is not detectable by the client.

4.2 Authentication of Clients using Digest Authentication

   Digest Authentication does not provide a strong authentication
   mechanism, when compared to public key based mechanisms, for example.

Franks, et al.   Standards Track   [Page 20]

RFC 2617   HTTP Authentication   June 1999

   However, it is significantly stronger than (e.g.) CRAM-MD5, which has
   been proposed for use with LDAP [10], POP and IMAP (see RFC 2195
   [9]). It is intended to replace the much weaker and even more
   dangerous Basic mechanism.

   Digest Authentication offers no confidentiality protection beyond
   protecting the actual password. All of the rest of the request and
   response are available to an eavesdropper.

   Digest Authentication offers only limited integrity protection for
   the messages in either direction. If qop=auth-int mechanism is used,
   those parts of the message used in the calculation of the WWW-
   Authenticate and Authorization header field response directive values
   (see section 3.2 above) are protected. Most header fields and their
   values could be modified as a part of a man-in-the-middle attack.

   Many needs for secure HTTP transactions cannot be met by Digest
   Authentication. For those needs TLS or SHTTP are more appropriate
   protocols. In particular Digest authentication cannot be used for any
   transaction requiring confidentiality protection. Nevertheless many
   functions remain for which Digest authentication is both useful and
   appropriate. Any service in present use that uses Basic should be
   switched to Digest as soon as practical.

4.3 Limited Use Nonce Values

   The Digest scheme uses a server-specified nonce to seed the
   generation of the request-digest value (as specified in section
   3.2.2.1 above). As shown in the example nonce in section 3.2.1, the
   server is free to construct the nonce such that it may only be used
   from a particular client, for a particular resource, for a limited
   period of time or number of uses, or any other restrictions. Doing
   so strengthens the protection provided against, for example, replay
   attacks (see 4.5). However, it should be noted that the method
   chosen for generating and checking the nonce also has performance and
   resource implications. For example, a server may choose to allow
   each nonce value to be used only once by maintaining a record of
   whether or not each recently issued nonce has been returned and
   sending a next-nonce directive in the Authentication-Info header
   field of every response. This protects against even an immediate
   replay attack, but has a high cost checking nonce values, and perhaps
   more important will cause authentication failures for any pipelined
   requests (presumably returning a stale nonce indication). Similarly,
   incorporating a request-specific element such as the Etag value for a
   resource limits the use of the nonce to that version of the resource
   and also defeats pipelining. Thus it may be useful to do so for
   methods with side effects but have unacceptable performance for those
   that do not.

Franks, et al.   Standards Track   [Page 21]

RFC 2617   HTTP Authentication   June 1999

4.4 Comparison of Digest with Basic Authentication

   Both Digest and Basic Authentication are very much on the weak end of
   the security strength spectrum. But a comparison between the two
   points out the utility, even necessity, of replacing Basic by Digest.

   The greatest threat to the type of transactions for which these
   protocols are used is network snooping. This kind of transaction
   might involve, for example, online access to a database whose use is
   restricted to paying subscribers. With Basic authentication an
   eavesdropper can obtain the password of the user. This not only
   permits him to access anything in the database, but, often worse,
   will permit access to anything else the user protects with the same
   password.

   By contrast, with Digest Authentication the eavesdropper only gets
   access to the transaction in question and not to the user's password.
   The information gained by the eavesdropper would permit a replay
   attack, but only with a request for the same document, and even that
   may be limited by the server's choice of nonce.

4.5 Replay Attacks

   A replay attack against Digest authentication would usually be
   pointless for a simple GET request since an eavesdropper would
   already have seen the only document he could obtain with a replay.
   This is because the URI of the requested document is digested in the
   client request and the server will only deliver that document. By
   contrast under Basic Authentication once the eavesdropper has the
   user's password, any document protected by that password is open to
   him.

   Thus, for some purposes, it is necessary to protect against replay
   attacks. A good Digest implementation can do this in various ways.
   The server created "nonce" value is implementation dependent, but if
   it contains a digest of the client IP, a time-stamp, the resource
   ETag, and a private server key (as recommended above) then a replay
   attack is not simple. An attacker must convince the server that the
   request is coming from a false IP address and must cause the server
   to deliver the document to an IP address different from the address
   to which it believes it is sending the document. An attack can only
   succeed in the period before the time-stamp expires. Digesting the
   client IP and time-stamp in the nonce permits an implementation which
   does not maintain state between transactions.

   For applications where no possibility of replay attack can be
   tolerated the server can use one-time nonce values which will not be
   honored for a second use. This requires the overhead of the server

Franks, et al.   Standards Track   [Page 22]

RFC 2617   HTTP Authentication   June 1999

   remembering which nonce values have been used until the nonce time-
   stamp (and hence the digest built with it) has expired, but it
   effectively protects against replay attacks.

   An implementation must give special attention to the possibility of
   replay attacks with POST and PUT requests. Unless the server employs
   one-time or otherwise limited-use nonces and/or insists on the use of
   the integrity protection of qop=auth-int, an attacker could replay
   valid credentials from a successful request with counterfeit form
   data or other message body. Even with the use of integrity protection
   most metadata in header fields is not protected. Proper nonce
   generation and checking provides some protection against replay of

previously used valid credentials, but see 4.8.

4.6 Weakness Created by Multiple Authentication Schemes

An HTTP/1.1 server may return multiple challenges with a 401
(Authenticate) response, and each challenge may use a different
auth-scheme. A user agent MUST choose to use the strongest auth-
scheme it understands and request credentials from the user based
upon that challenge.

Note that many browsers will only recognize Basic and will require
that it be the first auth-scheme presented. Servers should only
include Basic if it is minimally acceptable.

When the server offers choices of authentication schemes using the
WWW-Authenticate header, the strength of the resulting authentication
is only as good as that of the of the weakest of the authentication
schemes. See section 4.8 below for discussion of particular attack
scenarios that exploit multiple authentication schemes.

4.7 Online dictionary attacks

If the attacker can eavesdrop, then it can test any overheard
nonce/response pairs against a list of common words. Such a list is
usually much smaller than the total number of possible passwords. The
cost of computing the response for each password on the list is paid
once for each challenge.

The server can mitigate this attack by not allowing users to select
passwords that are in a dictionary.

Franks, et al.   Standards Track   [Page 23]

RFC 2617   HTTP Authentication   June 1999

4.8 Man in the Middle

Both Basic and Digest authentication are vulnerable to "man in the
middle" (MITM) attacks, for example, from a hostile or compromised
proxy. Clearly, this would present all the problems of eavesdropping.
But it also offers some additional opportunities to the attacker.

A possible man-in-the-middle attack would be to add a weak
authentication scheme to the set of choices, hoping that the client
will use one that exposes the user's credentials (e.g. password). For
this reason, the client should always use the strongest scheme that
it understands from the choices offered.

An even better MITM attack would be to remove all offered choices,
replacing them with a challenge that requests only Basic
authentication, then uses the cleartext credentials from the Basic
authentication to authenticate to the origin server using the
stronger scheme it requested. A particularly insidious way to mount
such a MITM attack would be to offer a "free" proxy caching service
to gullible users.

User agents should consider measures such as presenting a visual
indication at the time of the credentials request of what
authentication scheme is to be used, or remembering the strongest
authentication scheme ever requested by a server and produce a
warning message before using a weaker one. It might also be a good
idea for the user agent to be configured to demand Digest
authentication in general, or from specific sites.

Or, a hostile proxy might spoof the client into making a request the
attacker wanted rather than one the client wanted. Of course, this is
still much harder than a comparable attack against Basic
Authentication.

4.9 Chosen plaintext attacks

With Digest authentication, a MITM or a malicious server can
arbitrarily choose the nonce that the client will use to compute the
response. This is called a "chosen plaintext" attack. The ability to
choose the nonce is known to make cryptanalysis much easier [8].

However, no way to analyze the MD5 one-way function used by Digest
using chosen plaintext is currently known.

The countermeasure against this attack is for clients to be
configured to require the use of the optional "cnonce" directive;
this allows the client to vary the input to the hash in a way not
chosen by the attacker.

Franks, et al.   Standards Track   [Page 24]

RFC 2617   HTTP Authentication   June 1999

4.10 Precomputed dictionary attacks

With Digest authentication, if the attacker can execute a chosen
plaintext attack, the attacker can precompute the response for many
common words to a nonce of its choice, and store a dictionary of
(response, password) pairs. Such precomputation can often be done in
parallel on many machines. It can then use the chosen plaintext
attack to acquire a response corresponding to that challenge, and
just look up the password in the dictionary. Even if most passwords
are not in the dictionary, some might be. Since the attacker gets to
pick the challenge, the cost of computing the response for each
password on the list can be amortized over finding many passwords. A
dictionary with 100 million password/response pairs would take about
3.2 gigabytes of disk storage.

The countermeasure against this attack is to for clients to be
configured to require the use of the optional "cnonce" directive.

4.11 Batch brute force attacks

With Digest authentication, a MITM can execute a chosen plaintext
attack, and can gather responses from many users to the same nonce.
It can then find all the passwords within any subset of password
space that would generate one of the nonce/response pairs in a single
pass over that space. It also reduces the time to find the first
password by a factor equal to the number of nonce/response pairs
gathered. This search of the password space can often be done in
parallel on many machines, and even a single machine can search large
subsets of the password space very quickly -- reports exist of
searching all passwords with six or fewer letters in a few hours.

The countermeasure against this attack is to for clients to be
configured to require the use of the optional "cnonce" directive.

4.12 Spoofing by Counterfeit Servers

Basic Authentication is vulnerable to spoofing by counterfeit
servers. If a user can be led to believe that she is connecting to a
host containing information protected by a password she knows, when
in fact she is connecting to a hostile server, then the hostile
server can request a password, store it away for later use, and feign
an error. This type of attack is more difficult with Digest
Authentication -- but the client must know to demand that Digest
authentication be used, perhaps using some of the techniques
described above to counter "man-in-the-middle" attacks. Again, the
user can be helped in detecting this attack by a visual indication of
the authentication mechanism in use with appropriate guidance in

interpreting the implications of each scheme.

Franks, et al.   Standards Track   [Page 25]

RFC 2617   HTTP Authentication   June 1999

4.13 Storing passwords

     Digest authentication requires that the authenticating agent (usually
     the server) store some data derived from the user's name and password
     in a "password file" associated with a given realm. Normally this
     might contain pairs consisting of username and H(A1), where H(A1) is
     the digested value of the username, realm, and password as described
     above.

     The security implications of this are that if this password file is
     compromised, then an attacker gains immediate access to documents on
     the server using this realm. Unlike, say a standard UNIX password
     file, this information need not be decrypted in order to access
     documents in the server realm associated with this file. On the other
     hand, decryption, or more likely a brute force attack, would be
     necessary to obtain the user's password. This is the reason that the
     realm is part of the digested data stored in the password file. It
     means that if one Digest authentication password file is compromised,
     it does not automatically compromise others with the same username
     and password (though it does expose them to brute force attack).

     There are two important security consequences of this. First the
     password file must be protected as if it contained unencrypted
     passwords, because for the purpose of accessing documents in its
     realm, it effectively does.

     A second consequence of this is that the realm string should be
     unique among all realms which any single user is likely to use. In
     particular a realm string should include the name of the host doing
     the authentication. The inability of the client to authenticate the
     server is a weakness of Digest Authentication.

4.14 Summary

     By modern cryptographic standards Digest Authentication is weak. But
     for a large range of purposes it is valuable as a replacement for
     Basic Authentication. It remedies some, but not all, weaknesses of
     Basic Authentication. Its strength may vary depending on the
     implementation. In particular the structure of the nonce (which is
     dependent on the server implementation) may affect the ease of
     mounting a replay attack. A range of server options is appropriate
     since, for example, some implementations may be willing to accept the
     server overhead of one-time nonces or digests to eliminate the
     possibility of replay. Others may satisfied with a nonce like the one
     recommended above restricted to a single IP address and a single ETag
     or with a limited lifetime.

Franks, et al.   Standards Track   [Page 26]

RFC 2617   HTTP Authentication   June 1999

     The bottom line is that *any* compliant implementation will be
     relatively weak by cryptographic standards, but *any* compliant
     implementation will be far superior to Basic Authentication.

5 Sample implementation

     The following code implements the calculations of H(A1), H(A2),

```
    request-digest and response-digest, and a test program which computes
    the values used in the example of section 3.5. It uses the MD5
    implementation from RFC 1321.

    File "digcalc.h":

#define HASHLEN 16
typedef char HASH[HASHLEN];
#define HASHHEXLEN 32
typedef char HASHHEX[HASHHEXLEN+1];
#define IN
#define OUT

/* calculate H(A1) as per HTTP Digest spec */
void DigestCalcHA1(
    IN char * pszAlg,
    IN char * pszUserName,
    IN char * pszRealm,
    IN char * pszPassword,
    IN char * pszNonce,
    IN char * pszCNonce,
    OUT HASHHEX SessionKey
    );

/* calculate request-digest/response-digest as per HTTP Digest spec */
void DigestCalcResponse(
    IN HASHHEX HA1,            /* H(A1) */
    IN char * pszNonce,        /* nonce from server */
    IN char * pszNonceCount,   /* 8 hex digits */
    IN char * pszCNonce,       /* client nonce */
    IN char * pszQop,          /* qop-value: "", "auth", "auth-int" */
    IN char * pszMethod,       /* method from the request */
    IN char * pszDigestUri,    /* requested URL */
    IN HASHHEX HEntity,        /* H(entity body) if qop="auth-int" */
    OUT HASHHEX Response       /* request-digest or response-digest */
    );

File "digcalc.c":

#include <global.h>
#include <md5.h>
```

Franks, et al.   Standards Track   [Page 27]

RFC 2617   HTTP Authentication   -   June 1999


```
#include <string.h>
#include "digcalc.h"

void CvtHex(
    IN HASH Bin,
    OUT HASHHEX Hex
    )
{
    unsigned short i;
    unsigned char j;

    for (i = 0; i < HASHLEN; i++) {
        j = (Bin[i] >> 4) & 0xf;
        if (j <= 9)
            Hex[i*2] = (j + '0');
         else
            Hex[i*2] = (j + 'a' - 10);
        j = Bin[i] & 0xf;
        if (j <= 9)
            Hex[i*2+1] = (j + '0');
         else
            Hex[i*2+1] = (j + 'a' - 10);
    };
    Hex[HASHHEXLEN] = '\0';
};
```

```
/* calculate H(A1) as per spec */
void DigestCalcHA1(
    IN char * pszAlg,
    IN char * pszUserName,
    IN char * pszRealm,
    IN char * pszPassword,
    IN char * pszNonce,
    IN char * pszCNonce,
    OUT HASHHEX SessionKey
    )
{
    MD5_CTX Md5Ctx;
    HASH HA1;

    MD5Init(&Md5Ctx);
    MD5Update(&Md5Ctx, pszUserName, strlen(pszUserName));
    MD5Update(&Md5Ctx, ":", 1);
    MD5Update(&Md5Ctx, pszRealm, strlen(pszRealm));
    MD5Update(&Md5Ctx, ":", 1);
    MD5Update(&Md5Ctx, pszPassword, strlen(pszPassword));
    MD5Final(HA1, &Md5Ctx);
    if (stricmp(pszAlg, "md5-sess") == 0) {
```

```
Franks, et al.              Standards Track                    [Page 28]
```

```
RFC 2617                  HTTP Authentication              June 1999
```

```
        MD5Init(&Md5Ctx);
        MD5Update(&Md5Ctx, HA1, HASHLEN);
        MD5Update(&Md5Ctx, ":", 1);
        MD5Update(&Md5Ctx, pszNonce, strlen(pszNonce));
        MD5Update(&Md5Ctx, ":", 1);
        MD5Update(&Md5Ctx, pszCNonce, strlen(pszCNonce));
        MD5Final(HA1, &Md5Ctx);
    };
    CvtHex(HA1, SessionKey);
};

/* calculate request-digest/response-digest as per HTTP Digest spec */
void DigestCalcResponse(
    IN HASHHEX HA1,           /* H(A1) */
    IN char * pszNonce,       /* nonce from server */
    IN char * pszNonceCount,  /* 8 hex digits */
    IN char * pszCNonce,      /* client nonce */
    IN char * pszQop,         /* qop-value: "", "auth", "auth-int" */
    IN char * pszMethod,      /* method from the request */
    IN char * pszDigestUri,   /* requested URL */
    IN HASHHEX HEntity,       /* H(entity body) if qop="auth-int" */
    OUT HASHHEX Response      /* request-digest or response-digest */
    )
{
    MD5_CTX Md5Ctx;
    HASH HA2;
    HASH RespHash;
    HASHHEX HA2Hex;

    // calculate H(A2)
    MD5Init(&Md5Ctx);
    MD5Update(&Md5Ctx, pszMethod, strlen(pszMethod));
    MD5Update(&Md5Ctx, ":", 1);
    MD5Update(&Md5Ctx, pszDigestUri, strlen(pszDigestUri));
    if (stricmp(pszQop, "auth-int") == 0) {
        MD5Update(&Md5Ctx, ":", 1);
        MD5Update(&Md5Ctx, HEntity, HASHHEXLEN);
    };
    MD5Final(HA2, &Md5Ctx);
    CvtHex(HA2, HA2Hex);

    // calculate response
    MD5Init(&Md5Ctx);
    MD5Update(&Md5Ctx, HA1, HASHHEXLEN);
    MD5Update(&Md5Ctx, ":", 1);
    MD5Update(&Md5Ctx, pszNonce, strlen(pszNonce));
```

```
        MD5Update(&Md5Ctx, ":", 1);
        if (*pszQop) {
```

Franks, et al.   Standards Track   [Page 29]

RFC 2617   HTTP Authentication   June 1999


```
            MD5Update(&Md5Ctx, pszNonceCount, strlen(pszNonceCount));
            MD5Update(&Md5Ctx, ":", 1);
            MD5Update(&Md5Ctx, pszCNonce, strlen(pszCNonce));
            MD5Update(&Md5Ctx, ":", 1);
            MD5Update(&Md5Ctx, pszQop, strlen(pszQop));
            MD5Update(&Md5Ctx, ":", 1);
        };
        MD5Update(&Md5Ctx, HA2Hex, HASHHEXLEN);
        MD5Final(RespHash, &Md5Ctx);
        CvtHex(RespHash, Response);
};

File "digtest.c":


#include <stdio.h>
#include "digcalc.h"

void main(int argc, char ** argv) {

        char * pszNonce = "dcd98b7102dd2f0e8b11d0f600bfb0c093";
        char * pszCNonce = "0a4f113b";
        char * pszUser = "Mufasa";
        char * pszRealm = "testrealm@host.com";
        char * pszPass = "Circle Of Life";
        char * pszAlg = "md5";
        char szNonceCount[9] = "00000001";
        char * pszMethod = "GET";
        char * pszQop = "auth";
        char * pszURI = "/dir/index.html";
        HASHHEX HA1;
        HASHHEX HA2 = "";
        HASHHEX Response;

        DigestCalcHA1(pszAlg, pszUser, pszRealm, pszPass, pszNonce,
pszCNonce, HA1);
        DigestCalcResponse(HA1, pszNonce, szNonceCount, pszCNonce, pszQop,
          pszMethod, pszURI, HA2, Response);
        printf("Response = %s\n", Response);
};
```

Franks, et al.   Standards Track   [Page 30]

RFC 2617   HTTP Authentication   June 1999

6 Acknowledgments

   Eric W. Sink, of AbiSource, Inc., was one of the original authors
   before the specification underwent substantial revision.

In addition to the authors, valuable discussion instrumental in
creating this document has come from Peter J. Churchyard, Ned Freed,
and David M. Kristol.

Jim Gettys and Larry Masinter edited this document for update.

7 References

    [1]   Berners-Lee, T., Fielding, R. and H. Frystyk, "Hypertext
          Transfer Protocol -- HTTP/1.0", RFC 1945, May 1996.

    [2]   Fielding, R., Gettys, J., Mogul, J., Frysyk, H., Masinter, L.,
          Leach, P. and T. Berners-Lee, "Hypertext Transfer Protocol --
          HTTP/1.1", RFC 2616, June 1999.

    [3]   Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, April
          1992.

    [4]   Freed, N. and N. Borenstein. "Multipurpose Internet Mail
          Extensions (MIME) Part One: Format of Internet Message Bodies",
          RFC 2045, November 1996.

    [5]   Dierks, T. and C. Allen "The TLS Protocol, Version 1.0", RFC
          2246, January 1999.

    [6]   Franks, J., Hallam-Baker, P., Hostetler, J., Leach, P.,
          Luotonen, A., Sink, E. and L. Stewart, "An Extension to HTTP :
          Digest Access Authentication", RFC 2069, January 1997.

    [7]   Berners Lee, T, Fielding, R. and L. Masinter, "Uniform Resource
          Identifiers (URI): Generic Syntax", RFC 2396, August 1998.

    [8]   Kaliski, B.,Robshaw, M., "Message Authentication with MD5",
          CryptoBytes, Sping 1995, RSA Inc,
          (http://www.rsa.com/rsalabs/pubs/cryptobytes/spring95/md5.htm)

    [9]   Klensin, J., Catoe, R. and P. Krumviede, "IMAP/POP AUTHorize
          Extension for Simple Challenge/Response", RFC 2195, September
          1997.

    [10] Morgan, B., Alvestrand, H., Hodges, J., Wahl, M.,
          "Authentication Methods for LDAP", Work in Progress.

Franks, et al.   Standards Track   - [Page 31]

RFC 2617   HTTP Authentication   June 1999

8 Authors' Addresses

    John Franks
    Professor of Mathematics
    Department of Mathematics
    Northwestern University
    Evanston, IL 60208-2730, USA

    EMail: john@math.nwu.edu

    Phillip M. Hallam-Baker
    Principal Consultant
    Verisign Inc.
    301 Edgewater Place
    Suite 210
    Wakefield MA 01880, USA

    EMail: pbaker@verisign.com

Jeffery L. Hostetler
Software Craftsman
AbiSource, Inc.
6 Dunlap Court
Savoy, IL 61874

EMail: jeff@AbiSource.com

Scott D. Lawrence
Agranat Systems, Inc.
5 Clocktower Place, Suite 400
Maynard, MA 01754, USA

EMail: lawrence@agranat.com

Paul J. Leach
Microsoft Corporation
1 Microsoft Way
Redmond, WA 98052, USA

EMail: paulle@microsoft.com

Franks, et al.   Standards Track   [Page 32]

RFC 2617   HTTP Authentication   June 1999

Ari Luotonen
Member of Technical Staff
Netscape Communications Corporation
501 East Middlefield Road
Mountain View, CA 94043, USA

Lawrence C. Stewart
Open Market, Inc.
215 First Street
Cambridge, MA 02142, USA

EMail: stewart@OpenMarket.com

Franks, et al.   Standards Track   [Page 33]

RFC 2617   HTTP Authentication   June 1999

9. Full Copyright Statement

   Copyright (C) The Internet Society (1999). All Rights Reserved.

Franks, et al.   Standards Track   [Page 34]